Dealing with Non-Functional Requirements in Model-Driven Development

David Ameller, Xavier Franch
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
{dameller, franch}@essi.upc.edu

Jordi Cabot
INRIA-École des Mines de Nantes
Nantes, France
jordi.cabot@inria.fr

Abstract— **The impact of non-functional requirements (NFRs) over software systems has been widely documented. Consequently, cost-effective software production method shall provide means to integrate this type of requirements into the development process. In this vision paper we analyze this assumption over a particular type of software production paradigm: model-driven development (MDD). We report first the current state of MDD approaches with respect to NFRs and remark that, in general, NFRs are not addressed in MDD methods and processes, and we discuss the effects of this situation. Next, we outline a general framework that integrates NFRs into the core of the MDD process and provide a detailed comparison among all the MDD approaches considered. Last, we identify some research issues related to this framework.**

Keywords-non-functional requirements; model-driven development.

I. INTRODUCTION

Non-functional requirements (NFRs) are one of the main targets of research in the Requirements Engineering community [1] and their impact on practice has been documented in seminal papers [2], individual case studies [3] and types of industrial projects [4]. Given this reported impact of NFRs, we may say that any reliable and efficient software production process shall adequately handle them.

A software production paradigm that is gaining acceptance in the last years is Model-Driven Development (MDD) [5]. According to [6], "Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing". In other words, MDD uses models as the primary artifact of the software production process, and development steps consist of the (semi-)automated application of transformation steps over these models. Due to its promised benefits, MDD is being one of the main issues of communities and research groups like OMG, and is also mentioned as a driver in particular types of systems (e.g., [7] for self-adaptive systems).

According to the statement above, we may wonder whether current MDD approaches integrate NFRs in the production process. We will show in the paper that most current MDD approaches only focus on system functional requirements when generating system models, not integrating NFRs into the MDD process. Disregarding NFRs will usually provoke that the generated system does not completely satisfy some (if not all) of the stakeholders' expectations represented by NFRs. We believe that this is a strong argument against current MDD approaches that limit their success and applicability, and hampers their adoption by the industry.

In this vision paper, we are interested in identifying the challenges to overcome in order to effectively integrate NFRs into the MDD production process. To do so, we first provide more details about the current state of the art of MDD with respect to NFRs, understanding the limitations of the MDD methods that are not able to deal with NFRs, and analysing the approaches that apply some kind of treatment to NFRs. Next, we visualize a MDD general framework that smoothly integrates NFRs into the MDD process and discuss some variations. Last, we formulate some challenges and research lines stemming from this framework. To exemplify and motivate our findings, we use an academic exemplar about the development of a web portal for a travel agency.

II. BACKGROUND: MODEL-DRIVEN DEVELOPMENT

MDD is a development paradigm where models (and their transformation) play a fundamental role [5][6]. In MDD, models are used to specify, simulate, verify, test and generate the system to be built.

The most popular MDD method is the Model Driven Architecture approach, an OMG standard [8], that has been used as the basis for many other later MDD methods.

MDA distinguishes several types of models. Platform Independent Models (PIM) specify the software system in an independent way from the technology platform chosen to implement it. Platform Specific Models (PSM) refine the PIM to specificities of the implementation platform. That is, two different implementations of the same system would share the same PIM but have two different PSMs, each one adapted to the technological capabilities of each platform. A third type of model, Computation Independent Models (CIM, a kind of business model), exists, but in this paper, we will focus on the transformation from PIM to PSM.

Model-to-Model (M2M) transformations evolve a PIM into a PSM. Last, Model-to-Text (M2T) transformations are used to generate the executable system from the PSM. This step includes generating several code artifacts glued together: Java business classes, Oracle DB schemas, etc.

Fig. 1 summarizes the models and transformations considered in this paper.

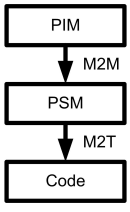

Figure 1. The MDA approach: models and transformations

III. MOTIVATION: THE TRAVEL AGENCY WEB PORTAL CASE

In this section we present an academic exemplar that we will use in the rest of the paper for illustration purposes.

The ACME travel agency offers transportation and accommodation services. The management has decided to deploy a web portal in order to offer some online functionalities to its customers, e.g: user management, payment facilities and searches (hotels, flights, etc.).

Together with these functionalities, many NFRs appear during the requirements elicitation process. E.g. since the portal is providing e-commerce transactions, security requirements like R1 = "The system shall detect and report unauthorised data accesses" are a must. The effect of this NFR can be manifold, for instance in a Web-based environment, firewalls are an architectural solution that supports this goal.

Other NFRs depend on the specific characteristics of the travel agency and the planned portal usage. For illustration purposes, let's consider two scenarios:

- *Scenario 1*. ACME is a specialized travel agency that offers luxury vacation packages to exotic destinations in 5-star hotels. It has a reduced portfolio of clients that plan their vacations using the system.
- *Scenario 2*. ACME is a world-wide leader travel agency. The company offers hundreds of packages that are assembled by combining data imported from other transportation and accommodation sites.

These scenarios impose some particular NFRs that capture their most essential characteristics. Thus, in Scenario 1, the number of expected visits is not too high and therefore scalability is not an issue. On the contrary, scalability and availability are key concerns to ensure the success of the portal in Scenario 2. Clearly, a good production process should be sensible to these differences and should result in different systems for each scenario. To make this statement more evident, let's consider one particular system dimension, the deployment architectural view as defined by Krutchen [9].

The deployment architectural view refers to the physical distribution of the software system components. Since the system we are considering as exemplar is a Web application, we may identify the following types of components [10]: the Web Server (WS), the Application Server (AS) and the Data Base Management System (DBMS). All these components can be deployed on the same node (Single Server Configuration, SSC), or using one of the several possible separations of components (e.g., separation of the DBMS). Also in the design of the deployment architecture it is possible to consider any type of component replication. Each deployment strategy affects some software quality attributes [11]. For instance, component replication (e.g. WS and AS) supports scalability, because more simultaneous connections may be established; replication also may improve efficiency especially if a load balancing component coordinates the incoming traffic. Table I sums up the effect of these strategies on some common architectural properties, according to [10].

At this point, the software architect has the duty of choosing the most adequate deployment strategy for the given set of NFRs, by comparing them with the effect of each strategy on the quality attributes. For the two scenarios described above, examples of convenient options are:

TABLE I. EFFECT OF COMPONENTS' DEPLOYMENT ON SOME ARCHITECTURAL PROPERTIES

	SSC	DBMS separated	DBMS & AS separated	Replication
Performance	Poor	Average	Good	Improve
Scalability	Poor	Poor	Poor	Improve
Availability	Poor	Poor	Poor	Improve
Maintenance	Good	Average	Average	Damage
Security	Poor	Good	Good	Neutral
Complexity	Good	Average	Poor	Damage

- For Scenario 1, the DBMS is kept separated from the WS and AS since scalability and availability are not major concerns, whilst security is increased by placing a firewall between the DBMS and the other two components (see Fig. 2, a). Replication is not implemented since its benefits are again concerning criteria that are not important for the given NFRs, whilst others would be damaged.
- For Scenario 2, since the agency provides a world-wide service, the WS and AS are replicated to improve availability and performance in those sites for which a greater number of clients may be expected. A load balancing system coordinates the different WS to improve performance even more. DBMS containing data local to the sites are put together with the WS and AS, and firewalls are also deployed for protecting each local DBMS. As a final decision, a centralized DBMS contains some replicated data that may be of interest for performing some data mining operations. Fig. 2, (b), provides the whole picture.

Other deployment options are possible. It is not a goal of this section to discuss them, but just to emphasise the fact that the final form of the software architecture depends on the set of elicited NFRs and to give some initial idea of the type of knowledge to manage and decisions to be made.

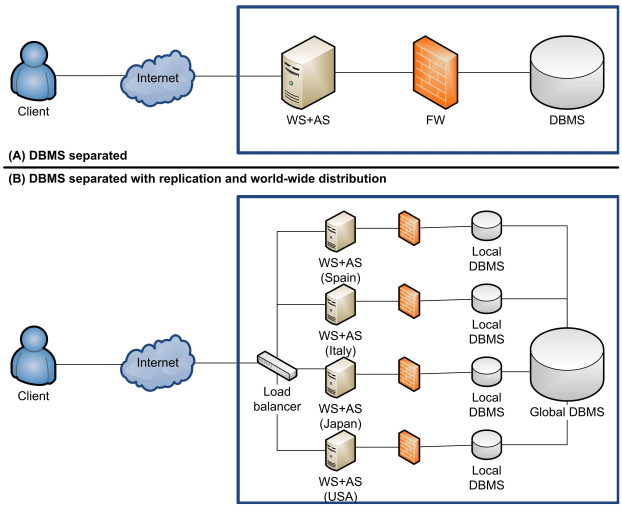

Figure 2. Two different deployment architectures for the Web portal case.

IV. NON-FUNCTIONAL REQUIREMENTS IN MODEL-DRIVEN DEVELOPMENT: STATE OF THE ART

In the previous section, we have shown that NFRs have an important effect in the final form that the software system takes. If we consider MDD, we may say that an optimal MDD production process should be able to deal with the set of elicited NFRs and use them to select and apply the most adequate transformations, in order to generate a software system that satisfies the desired NFRs. In this section we investigate to what extent this need is currently fulfilled.

We distinguish MDD aproaches that do not consider NFRs as part of the transformations, from those that do.

A. MDD Approaches not supporting NFRs

We may find a great variety of MDD-based approaches in the literature, many of them following the two-level (PIM and PSM) classification introduced in the OMG's MDA approach [8]. Among the most popular ones, we find the Executable UML proposals, with [12] as the most popular representative. Executable UML methods use a reduced subset of UML that it is directly executable, either using UML interpreters or by providing a direct translation from the models to the final code.

Using such Executable UML methods, the travel agency model consists of use case diagrams, class diagrams, sequence diagrams and activity diagrams that express the roles, functionalities, data and behaviour of the system. None of these artifacts is able to express any kind of NFR. Thus, the transformation from PIM to PSM is fixed and it is not possible to choose the most appropriate strategy for a given set of NFRs: the PSM will be close to, or far from, the elicited NFRs depending on the system quality factors implicitly encoded in the predefined transformations. Some action is required in order to make the MDD approach effective.

We believe that this a critical situation, even more considering that this Executable UML method [12] is the basis for the upcoming OMG standard "*Semantics of a Foundational Subset for Executable UML Models*" that pretends to increase the use of UML in a MDD context.

If we consider the general form of MDD (see Fig. 1), we may envisage two different, non-exclusive approaches to make a generated product compliant with the stated NFRs:

1. The software developer directly modifies by hand the result of the MDD process (see Fig. 3, left). In its simplest form, she directly modifies the code obtained after the final M2T transformation. In the best case, she will able to work at the PSM level, modifying the model to adapt it to the NFRs, and then use the M2T transformation (possibly modified somehow) to generate the code. This manual adaptation of the system collides with the essence of the MDD paradigm and has several drawbacks:
   - Takes longer to produce the software.
   - Provokes lower reliability of the final product due to the human-based post-process.
   - Damages traceability and thus comprehension.
   - In case of changes due to maintenance, either the post-process has to be replicated or the maintenance is directly made on the final product.

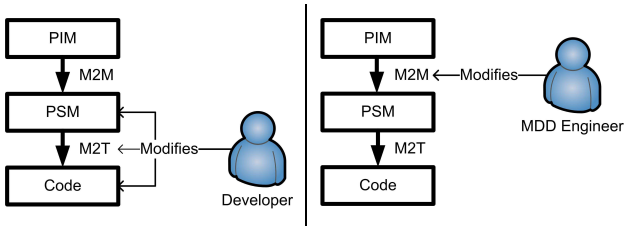

Figure 3. Dealing with NFRs in a classical MDD approach

2. The MDD engineer modifies the M2M transformation in order to obtain a PSM that satisfies the NFRs (see Fig. 3, right). In our example above, we could have three transformations for producing PSM compliant to the *SSC*, *DBMS separated*, and *DBMS and AS separated*, strategies. The drawbacks above are therefore solved, but others appear in their place:
   - The complexity of the MDD framework is greater, because there are more transformations to maintain.
   - It is difficult to anticipate all the possible scenarios, in fact it may be even impossible (e.g., in Table I, replication may be applied in many different ways, and each would require a different transformation).
   - The selection of the most appropriate transformation (for the given set of NFRs) to apply relies on the software architect, becoming a human-based pre-process, incrementing thus the likelihood of errors in decision-making.
   - When the software architect realizes that the available transformations are not adequate for the current process it is necessary to build a new ad-hoc one, making the initial configuration time longer.

The two approaches presented above represent two extreme cases. Hybrid solutions may also exist, where some NFRs are addressed by the M2M transformation and others remain under the final responsibility of the developer.

To sum up, we may state that MDD approaches that are not able to deal with NFRs in the software production process suffer from severe drawbacks that must be manually fixed by either the developer or the MDD engineer and that, therefore, may compromise their adoption.

The situation is even worse when considering not the theory but the real state of practice of MDD, hampered by the limitations of MDD tools available in the market. For instance, their code-generation capabilities are limited to particular technologies/languages (which implies that usually only some parts of the system can be transformed and generated by the tool) and it is not always possible to change the predefined M2M and M2T transformations offered by the tool. Therefore, a scenario more realistic than those depicted in Fig. 3 is described below (see Fig. 4):

- The MDD engineer specifies a PIM that contains only information about system functional aspects.
- The software architect defines (or chooses from the modeling tool she is using) a set of transformations that are applied to different parts of the PIM, generating each an unrelated part of the target PSM. Each generated PSM part is compliant with a particular technology.

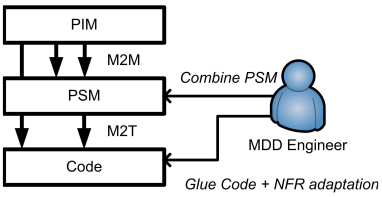

Figure 4. Dealing with NFRs using current MDD technologies

- M2T transformations are applied to the PSM for obtaining the final code.
- The developer complements the generated code and combines the generated code-excerpts into a coherent architecture.

This process is adding some new drawbacks:

- There is not a single transformation generating a complete PSM, but a set of partial transformations generating separated pieces that may yield an incomplete PSM. Even, some tools skip the generation of the PSM and jump directly to the code.
- The different pieces generated by the transformations need to be manually linked, writing additional glue code.
- With respect to NFRs, each transformation results on PSM parts that may not satisfy the stated NFRs (in fact, depending on the available transformations each excerpt can enforce different and maybe contradictory NFRs).

B. *MDD approaches that deal with NFRs*

To know about the approaches that currently deal with NFRs in the MDD process, we have set up a Systematic Literature Review [13] that we briefly describe belos. Concretely, we have search in the Web of Science (WoS) by topic (title+abstract+keywords) using as search string:

("model driven" or "model-driven" or "MDD" or "MDA" or "MDE") and ("non functional" or "nonfunctional" or "non-functional" or "quality" or "NFR") and ("requirements" or "aspects" or "properties")

From that search we obtained 228 results, reduced to 36 after reading the title and the source of each publication (remarkably conference name), then reduced to 15 when reading the abstracts, and finally, to 11 representative papers after reading the full text. To complement the results from WoS we have also analyzed 26 additional papers that are cited by this 11 and didn't appear in the WoS-based search, and from this analysis 2 more papers were selected.

All the analyzed approaches focus on a particular MDD activity and/or type of NFR. Concerning analysis, they either focus on the modeling of the NFRs, on their use as part of a model transformation or on their analysis. Concerning types of NFR, most approaches are centered in only one or two NFRs and/or for a specific domain (see Table IITABLE II.). In what follows we provide some additional details.

1) Modeling NFRs. Several authors propose to model NFRs using UML extensions [14][15][16], including the OMG standard UML profiles MARTE [17] and QoS-Profile [18]. Others designed a specific metamodel to represent NFRs [19][20][21].

TABLE II. APPROACHES THAT DEAL WITH NFRS IN MDD ACCORDING TO OUR SYSTEMATIC LITERATURE REVIEW

Ref.	Type of NFR addressed	Domain	Instrument
Modeling NFRs			
[14]	Operationalizable NFRs	Independent	NFR Framework + UML annotations
[15]	Security, Fault Tolerance	SOA	UML Profile
[16]	Any	Independent	UML Profile
[19]	Performance	SOA	Own metamodel
[20]	Resource Usage	Embedded systems	Own metamodel
[21]	Usability	Web IS	Own metamodel
Model Transformation			
[22]	Quality of Service (QoS)	Independent	Patterns
[23]	Any	Independent	Patterns
Model Analysis			
[24]	Quality of Service (QoS)	Independent	Measurable models
[25]	Performance, Reliability	Independent	Markov models
[26]	Performance, Reliability	SOA	Probabilistic models
[27]	Reliability	Independent	LTSA*
[28]	Any	Independent	Not specified

** Labelled Transition Systems Analyser*

2) *Model transformation*. Given a set of NFRs, [22] proposes a set of patterns that satisfy QoS requirements. In [23], the proposed patterns consider architectural aspects.

3) *Model analysis*. Following the ideas proposed in [24], these proposals analyze the satisfaction of a given NFR in a particular software design by transforming this design into a specific formalism (different for each NFR) in which the analysis can take place. Examples are [25][19][26][27]. In these approaches each kind of NFR may be seen as a whole dimension of the software. [28][25] propose analyzing each NFR type separately and also to use different abstraction levels for NFRs (at CIM, PIM and PSM levels).

As a conclusion, we may say that although several valuable approaches have been proposed that deal with NFRs in the MDD process, none of them propose an integrated view, which is the goal of this vision paper.

V. NON-FUNCTIONAL REQUIREMENTS AS PART OF THE MODEL-DRIVEN DEVELOPMENT PROCESS

In the previous section we have shown that MDD approaches that do not consider NFRs as part of the generation process suffer from serious drawbacks, and that, unfortunately, this is the predominant type of approach nowadays. In this section we discuss a general solution to this problem.

A. *Basic concepts for dealing with NFRs in MDD*

Many authors have reported the intimate relationship among requirements and architectures and also the great impact that NFR have on architectures [29][30][31]. For example, in the analysis of Section IV, we have shown how new components (e.g., firewalls and load balancers) and physical component allocation (e.g., replication) can be justified in terms of the NFRs that must be satisfied. Therefore, we envisage an approach to MDD in which the PIM is transformed into a complete software architecture. Transformations have the mission of allocating the responsibilities coming from the PIM functional part to components that are deployed into an architecture that satisfies the NFRs.

But NFRs are also important when determining the choice of technologies needed to implement the architecture. For instance, it may be necessary not just to know that a relational data base is needed, but also that a particular brand, or even version and release, is the right choice. Interoperability requirements (e.g., "The portal shall be compatible with our current data base in the central management system") or non-technical requirements [32] (e.g., "The data base vendor shall provide 24×7 call center assistance") are clear examples of NFRs with this effect.

Table III describes the main elements that constitute our envisioned framework proposal. Remarkably, and following the discussion above, we introduce two kinds of models between the PIM and the code: the model representing the architecture, and the model representing the technological solution. Whilst the latter is clearly a PSM, the former lays in between the two levels of abstraction and therefore we denote it by *PIM/PSM*. For each kind of model, we include between parentheses the requirements that are satisfied by the elements in that model,. Finally, as a consequence of having two different intermediate models among the PIM and the code, we have two corresponding M2M transformations, $M2M_{arch}$ and $M2M_{tech}$.

B. An NFR-aware MDD process: Integrating NFRs into the PIM

We believe that the most natural way to integrate NFRs into the MDD process is by considering NFRs from the very beginning of the development process, i.e. as part of the PIM. As functional requirements, NFRs become first-order citizens of the MDD process.

The MDD process then works as follows:

- The analyst specifies a PIM that contains both the functional and non-functional requirements, PIM(*f*+*nf*).
- The MDD decisional engine decides, given the PIM(*f*+*nf*) and the contains of the MDD knowledge base (i.e., information about non-functionality, architectures and technologies), the final form of the transformation $M2M_{arch}$:

$$M2M_{arch}: PIM(f+nf) \rightarrow PIM/PSM(f+nf_0)$$

  This transformation takes PIM(*f*+*nf*) as input and produces PIM/PSM(*f*+nf_0), a model describing an architecture that implements all the functionality *f* in a way that satisfies the elicited subset of NFRs nf_0 whose satisfaction depends on the decisions made at the architectural level.
- Once the PIM/PSM(*f*+nf_0) has been generated, the MDD decisional engine applies a second M2M transformation that generates the PSM for the desired final implementation technology. This PSM follows the architectural guidelines expressed above (and thus, satisfies nf_0) but also takes into account all the remaining *nf* (directly related to technologies, as those mentioned in V.A), forcing the adoption of a particular technology or product:

$$M2M_{tech}: PIM/PSM(f+nf_0) \rightarrow PSM(f+nf)$$

- Last, a simple M2T transformation can be applied to obtain the code from the technology:

$$M2T: PSM(f+nf) \rightarrow Code(f+nf)$$

In the framework, the transformations are presented as single functions. In fact, this is a simplified view since a transformation will be in fact a composition of the application of many transformation rules. Thus, we may say that:

TABLE III. CONCEPTS NEEDED WHEN INTEGRATING NFRS INTO THE MDD PROCESS

Concept	Definition	Example
f, nf	The elicited functionality and non-functionality of the system (not represented as model)	An IEEE 830-compliant Software Requirements Document
PIM(f)	PIM that specifies some functionality *f* of the system	A UML class diagram specifying the system data
PIM(f+nf)	PIM that specifies all the requirements of the system	An *i** model of the system complemented with UML data and behavioural diagrams
PIM/PSM(f+nf)	Model mixing PIM and PSM levels that specifies some functionality *f* satisfying the NFRs *nf*	A 3-layer architecture expressed with the ACME Architectural Description Language (ADL)
PSM(f+nf)	PSM that specifies some functionality *f* satisfying the NFRs *nf*	A model with a class diagram annotated with database stereotypes (e.g. <<PK>>, <<Table>>) that only have meaning for the Oracle DBMS
Code(f+nf)	Executable system that implements the functionality *f* satisfying the NFRs *nf*	Implementation of the 3-layer architecture above using Java components, XML interchange data formats, Oracle DB schema, etc.
M2M	M2M transformation from a PIM to a PSM	Transformation of a UML specification into a technological solution including an Oracle data base and a Pound load balancer, among others
$M2M_{arch}$	M2M transformation from a PIM to a PIM/PSM that represents the architecture of the system	A mapping from an Executable UML model of functionality into a 3-layer architecture expressed with the ACME ADL
$M2M_{tech}$	M2M transformation from a PIM/PSM into a PSM that represents the technological solution of the system	Transformation of the ACME architectural model into a representation of technology that, for instance, annotates a class diagram with Oracle-compliant database stereotypes
M2T	M2T transformation from a PSM to the executable system	Transformation of a stereotyped UML diagram to EJB Java classes

$$M2M(m) = r_{k_{m2m}}(\ldots(r1_{m2m}(m)\ldots)$$

being M2M either $M2M_{arch}$ or $M2M_{tech}$. From a conceptual point of view, the vision of the transformation as a single function is a convenient simplification that does not hamper the generality of the approach.

C. *Example: deciding the need of firewall components*

In this example we illustrate the kind of information to record, and steps to apply, in order to derive part of the architectural model for the Web portal example presented in Section III. We remark that the notations used to represent the models, and even the concrete steps taken and their order are just an example of how they may look like, we refer to Section VI for further discussion.

We distinguish three parts: the knowledge base used by the MDD decisional engine; the creation of the starting PIM; and the application of our MDD process itself. For the latest, we will restrict to the creation of the PIM/PSM.

1) Representing the MDD knowledge. We focus on the concepts directly related to NFRs. First, it is necessary to represent the types of NFRs managed and the consequences that architectural decisions may have on them. We can represent this using a tabular structure (like Table I) or by means of a notation like the NFR framework [30], used with similar purposes in several works (e.g., [33][34]). The model depicted in Fig. 5, top, shows an excerpt of the information needed, with several softgoals to represent the NFRs and two particular operationalizations for them (each with a different positive/negative effect on them).

Next, it is necessary to represent the implications of each operationalization on the architecture. This is described for the firewall case in the lower part of Fig. 5. The firewall solution requires three participants: the firewall component itself, and two subsystems that are connected, the internal (i.e., protected) and the external ones. These elements are in fact instances of architectural metaelements, e.g. subsystem, defined according to some architectural ontology like those in [9][35].

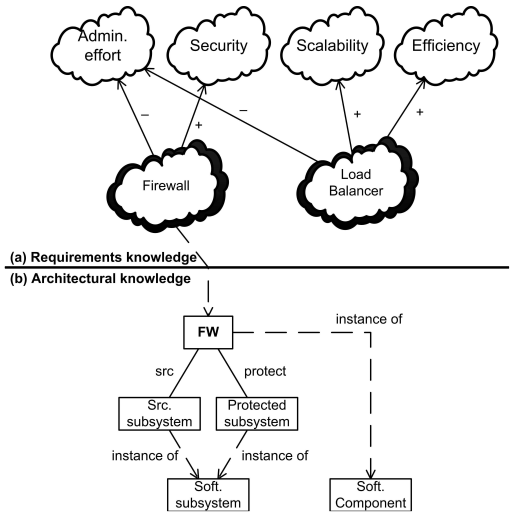

Figure 5. Knowledge representation in the MDD Knowledge Base

*2) Creating the PIM(*f+nf*)*. The process starts with the PIM definition. For the functional part we can still follow any existing proposal, e.g. Executable UML. For the NFRs, we may decide to use a natural language representation based on requirement patterns as in [36]. which allows to establish easily the link between such NFRs and the predefined NFRs types in the MDD knowledge base (KB). For instance, Fig. 6 represents the R1 NFR (see Section 3) and the link with the *Security* NFR type maintained in the KB.

3) Creating the PIM/PSM. The following actions are taken to process R1:

- The MDD decisional engine chooses, using some appropriate analysis technique (e.g., [30][37]), the Firewall operationalization to support R1.
- As a consequence of the system being a Web application (which is a decision coming from the intrinsic nature of a Web portal), a transformational step decomposes the system into three main subsystems: WS, AS and DBMS. The MDD Knowledge Base knows that the communication between these subsystems is: WS ↔ AS ↔ DBMS.
- The assignment of elements from the functional part of PIM(*f+nf*) into WS, AS and DBMS, takes place. In particular, the data model elements are assigned into DBMS.
- Since R1 is referring to data protection, and since DBMS is bound to data, the MDD decisional engine decides that the protected subsystem for the firewall is the DBMS. Since the communication for Web application is from AS to DBMS, it is also possible to deduce that the "source" of the Firewall is the AS.
- In Scenario 1, since there is no replication, there are just one AS and one DBMS, and thus just one Firewall is induced (see Fig. 6). In Scenario 2, due to replication, there are as many Firewalls as pairs AS-DBMS. The fact that the WS and the AS are deployed together completes the information needed to determine the final form of the architecture.

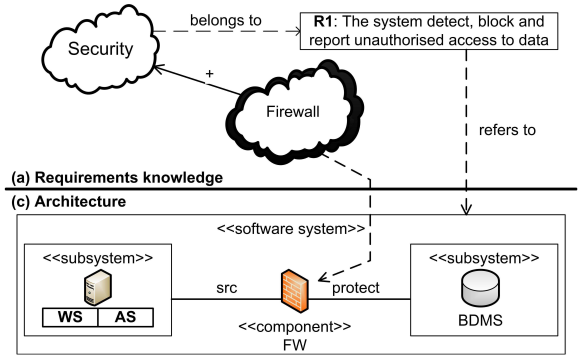

Figure 6. Architectural consequences of R1 in the NFR-aware MDD process

D. *An NFR-aware MDD process: Using NFRs for decision-making*

Although the framework presented above is theoretically neat, it is clear that it has a lot of complexity to manage. Remarkable, it requires:

- To determine the most adequate formalism for representing the non-functional part of PIM(f+nf). We have used in the example the NFR framework, that is basically a qualitative-oriented one, but also more quantitative approaches may be considered, e.g. in QoS-style [38].
- To embody in the MDD decisional engine all the knowledge needed to make informed architectural decisions, i.e. to determine the concrete form that the M2M functions take. In other words, the M2M are required to provide a correct output in all possible situations. This is a very strong condition mainly because of: 1) the amount of knowledge to represent is huge and not always clear; 2) the conflicting nature of NFRs: architectural decisions permanently require trade-off analysis.

These problems lead to propose a second alternative specially interesting until clearly accepted technical solutions for the previous points are provided. Instead of considering NFRs as part of the PIM and then be an input of the MDD process, we may consider that the MDD process asks the software architect the NFR-related information as it is needed. The resulting process becomes:

- The analyst specifies a PIM that contains just the functional requirements, PIM(f).
- The transformation function $M2M_{arch}$ takes PIM(f) as input and produces PIM/PSM(f+nf_0) (nf_0 stands again for those NFRs that concern the architecture). To produce this output, the MDD process presents a series of questions $Q = \{q_1, ..., q_n\}$ to the software architect whose answer is needed in order to decide the transformation steps to apply. The software architect provides answers $A = \{a_1, ..., a_n\}$ according to the NFRs nf_0. If we denote by σ_{arch} the function that records the mapping from questions to answers, $\sigma_{arch}(q_i)=a_i$, the transformation function is defined as:

$$M2M_{arch}: PIM(f) \times \sigma_{arch} \rightarrow PIM/PSM\,(f+nf_0)$$

- The subsequent M2M transformation for the technology acts the same, requiring a similar σ_{tech} function to obtain from the MDD engineer the information needed to make informed decisions:

$$M2M_{tech}: PIM/PSM\,(f+nf_0) \times \sigma_{tech} \rightarrow PSM\,(f+nf)$$

- The M2T transformation is not affected:

$$M2T: PSM(f+nf) \rightarrow Code(f+nf)$$

Questions that the MDD decisional engine may raise to the architect may be manifold. For instance, there may be high-level questions like the type of organization with respect to departments (e.g., to decide which nodes are part of the physical architecture) and lower-level ones like the probability of execution of a given operation or use case.

The two NFR-aware approaches presented in this Section V represent two extreme visions but of course we can think of hybrid solutions, in which the MDD decisional engine supports decision-making for some types of NFRs, architectural elements and technologies, whilst the software architect and developer may provide the information missing under demand.

E. Comparison

In this section we compare the two NFR-aware approaches presented in this section with the three approaches presented in Section IV. Fig. 7 aligns the five approaches for an easier comparison. When comparing, please pay attention to: the number and nature of models and transformations; the extent of requirements in the models (enclosed in parenthesis); and the type of interaction with the human assistant (where, and in which direction). Table IV includes a detailed comparison respect to several criteria.

In short, the main benefits of NFR-aware approaches are:

- NFR-aware approaches fully integrate NFRs into the MDD process. Especially in the first NFR-aware framework presented (Fig. 7(d)), NFRs are considered at the same level than the functional specification, being both part of the departing PIM. Knowledge may be incrementally stored in the MDD knowledge base (gradually improving accuracy of results) and may be reused in each new project.
- As a consequence, there is no need for the developer neither to write glue code (since the different components of the PSM model are already interrelated) nor to adapt the code to satisfy the NFRs (since the NFRs have been already taken into account when creating the PSM model).
- Instead of obtaining several incomplete PSM, using a single transformation that targets a specific architecture a single, a comprehensive and unified representation of the system is derived.
- Two levels of abstraction are recognized, one for representing architectures, other for representing technologies. This distinction fits with the levels of abstraction that practitioners use in their daily work.
- The explicit representation of NFRs allows defining model transformation repositories inside the MDD knowledge base that can be used to select the proper transformations to apply. Also, when NFRs are considered at the PIM level, classical analysis techniques from Requirements Engineering may be applied in the early stages of the MDD process.
- Hybrid approaches (between options from Fig. 7(d) and 7(e)) allow customizing the NFR-awareness to the resources, skills and preferences of software architects. For instance, an empirical study that we recently conducted shown that software architects are reluctant to lose all the control over the architectural decisions to be made [39].

But as the Table IV shows, these benefits are not for free. Incorporating NFRs results in higher modeling effort, both for constructing the PIM and for building the MDD knowledge base. Also, it requires discipline to keep this MDD knowledge base up to date. Complexity of the MDD process is the overall challenge to overcome.

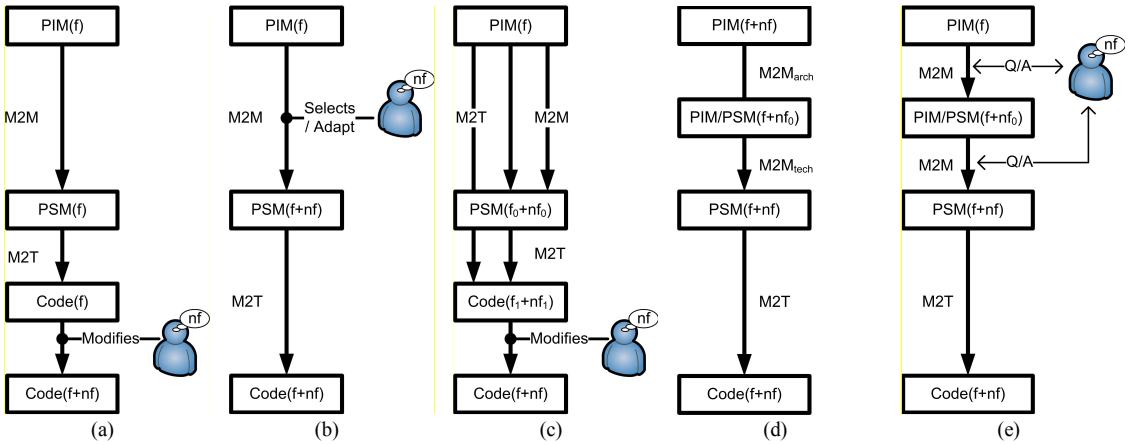

Figure 7. The five different possibilities: (a) manual modification of the code; (b) manual configuration of the transformation; (c) multiple and heterogeneous transformations; (d) integrating NFRs into the PIM; (e) elicitating NFR-related information from human. It holds that $nf_0 \subseteq nf_1 \subseteq nf$.

TABLE IV. COMPARISON AMONG THE DIFFERENT MDD STRATEGIES ANALYSED IN THE PAPER

	MDD approaches not dealing with NFR			NFR-aware MDD frameworks	
	Fig. 7(a)	Fig. 7(b)	Fig. 7(c)	Fig. 7(d)	Fig. 7(e)
Project set-up					
Modeling time	Fair (just functionality is modeled)	Fair (just functionality is modeled)	High (several notations used to build the PIM)	Very high (NFRs need to be modeled)	Fair (just functionality is modeled)
MDD configuration time for a particular project	None (transformation applied as is)	Probably high (if a new transformation is needed)	None (transformations applied as are)	None (transformations applied as are)	None (transformations applied as are)
Production process					
Production time once configuration finished	High (full post-process adaptation)	Fair (slight post-process adaptation will probably be needed)	Very high (post-process adaptation and gluing)	None (if transformations are complete)	Low (guided conversation with human)
Criticality of human intervention during production	High (high responsibility of the developer at the end)	Fair (slight post-process adaptation will probably be needed)	High (high responsibility of the developer at the end)	None (since there are no human interactions)	Low (she just needs to respond to very concrete questions)
Complexity of the process	Low (the MDD infrastructure is static)	High (several transformations co-exist)	Very high (several heterogeneous transformations exist)	High (the transformations used will be more complex)	Moderate (human intervention simplifies the process)
Knowledge reuse and learning ability	Very low (just the functional-related knowledge is reused)	Low (learning ability comes from the MDD engineer)	Very low (just the functional-related knowledge is reused)	Very high (NFR-related knowledge may be reused and may grow)	High (some NFR-related knowledge may be reused and may grow)
MDD KB maintenance cost	Low (since it just covers functionality)	Fair (updates up to the MDD engineer)	Fair (updates up to the MDD engineer)	Very high (all new knowledge needs to be modeled)	High (some new knowledge needs to be modeled)
Product management					
Product Traceability	Very low (generated product modified)	Fair (depending on the complexity of the post-process adaptation)	Extremely low (generated product modified; information across models)	Potentially complete (all decisions can be traced)	High (answers to questions may be recorded)
Maintainability	Very low (changes made are probably lost if product generated again)	Fair (depending on the complexity of the post-process adaptation)	Very low (changes made are probably lost if product generated again)	Very high (it is possible to work only at PIM level)	High (functionality at PIM level; changes on NFRs require new questions)

VI. DISCUSSION AND RESEARCH AGENDA

Putting a NFR-aware MDD production process into practice looks like a great challenge. In this section we outline the most relevant issues to investigate with emphasis on requirements-related issues.

1) Modelling of NFRs at the PIM-level. (a) Which types of NFRs are most relevant to the MDD process? It is important to devote efforts to the NFRs that software architects perceive as the most important. Surveys (e.g., [39]) and interviews are needed. (b) Which notation to use for representing NFRs? As comented, quantitative and

qualitative approaches are the two (non-exclusive) big families. This is an old research topic in Requirements Engineering (already appearing in the 2000's roadmap [40]) and results obtained in contexts other than MDD may be transferred here. (c) How NFRs may be linked to the functional elements? Some approaches have been cited [14][17][18] at this respect.

2) Elicitation and representation of architectural knowledge. (a) Which types of architectural knowledge exist and how are they used in practice? Again empirical studies are needed to give light to this question [41]. (b) Which are the quality attributes corresponding to these styles? (c) Which are the matching rules that allow determining the architectural solution that best fits the required NFRs?

3) Nature of models. The classification of MDD models into CIM, PIM and PSM as defined in the MDA approach results in some rigidity in our context. We have already defined the architectural model as an intermediate PIM/PSM model. The situation may even be more confusing if we inject the concept of architectural view [9] into the core of the MDD process. For instance, we may envisage that the evolution from the PIM down to the architectural models yields to a sequence of models in decreasing abstraction level, corresponding to the different architectural views, from the logical view down to the physical view. In this case, labelling the models may be artificial. We remark too that current MDD approaches focus on the architectural logical view, thus addressing other views is a progress by itself.

4) M2M transformations. Challenges are: (a) Gradually developing and incorporating in the framework transformations for all popular architectural styles. (b) Selecting the best alternative for each non-deterministic transformation depending on the expected NFRs. (c) Defining a transformation composition language for gluing separate transformations into the MDD models. This last point is highly connected with the vision promoted in [24][27] where different types of NFR are handled separately. Being true that the specifities of each NFR type makes it difficult to treat them uniformily, it is also clear that we need to be able to reconcile them since the generated system needs to fulfil all of them together. (d) The framework presented here conceives the application of transformation (and thus obtention of models) top-down with respect to abstraction level. However, this does not need to be always this way. For instance, a technological NFR fixing the brand and release of the data base product will have an implicit consequence on some other more abstract model, namely to know that a data base of a particular type (relational, OOR, …) has to be integrated into e.g. the development view of the architecture.

5) The MDD core: decisional engine and knowledge base. The research agenda includes: (a) Being able to keep and reuse the knowledge acquired in MDD projects (e.g., success and failure experiences). (b) Exploring the applicability of artificial intelligence techniques for taking informed decisions (case-based reasoning, Bayesian networks, etc.). (c) Exploit the knowledge of software architects to improve the automation of the process by means of a comprehensive program of interviews and surveys. (d) Define the roles and responsibilities that play a part in the MDD process: software architect, MDD engineer, software developer, domain expert, etc.

6) Variations from the proposed frameworks. Being the presented frameworks general, variations may be thought to be formulated. Let's consider one particular variation, namely the incorporation into the MDD process of the concept of architectural style. According to [42][43], an architectural style consists of the description of the types of architectural components supported, their organization, and how can they be integrated. In some sense, we may say that different architectural styles use different ontologies, e.g. whilst SOA talks about services, choreography and MOM, layered architectures introduce layers, persistence and push communication model. Incorporating this concept into the framework has consequences on its very core. If the M2M translation from PIM to PIM/PSM renders a software architecture, it follows that each architectural style requires a different metamodel, thus both PIM/PSM models and M2M transformations are dependant on the architectural style, becoming families of models and functions:

$$\left(\mathrm{M2M}_{\mathrm{arch}[st]}\colon \mathrm{PIM}(f{+}nf) \rightarrow \mathrm{PIM/PSM}_{[st]}\,(f{+}nf_0) \right)_{st \in \mathrm{style}}$$

Determining the architectural style should be the first decision to be made in the MDD process. Adopting a pure MDD perspective, it should be determined from the PIM(f+nf). However, it is true that the decision of whether it must be, for example, an SOA or a Web rich client architecture is often a decision made before the MDD process starts for reasons that are not always tangible and are only in the architect's mind.

7) Correctness and completeness issues. Last but not least, we mention the need of accurately investigating the notion of correctness of an NFR-aware approach. We may envisage the following conditions that need to be refined to the chosen formalisms. A couple of examples of predicates to investigate are:

- The NFRs should be correct both independently (e.g., there are not contradictory NFRs) and when referred to the functionality f (each functional element is qualified by meaningful types of NFRs): $correct(nf) \wedge applicable(nf, f)$.
- The knowledge embedded in the MDD knowledge base should find feasible alternatives for any given NFRs that fulfil the correctness and applicability conditions above: $correct(nf) \wedge applicable(nf, f) \Rightarrow \mathrm{reducible}(KB, nf)$

VII. CONCLUSIONS

In this vision paper we have: explored the state of the art; envisaged some generic solution to the identified problems; and enumerated new lines of research and challenges to overcome; of one requirement-related practice, the management of non-functional requirements (NFR) in the model-driven development (MDD) production paradigm.

Being this a vision paper, the main goal has been to agree on a perspective of the current state of the addressed problem and in the need to keep progressing towards several directions. Concerning the state of the art:

- We have analysed how MDD methods not dealing with NFRs behave to ensure their satisfaction.
- We have run a systematic literature review to learn insights of the MDD methods that deal with NFRs.

Concerning the improvement of this state of the art:

- We have formulated an NFR-aware general framework which allows customization to different settings with their own peculiarities
- We have discussed variations on this framework.
- We have aligned and thoroughly compared the different alternatives discovered, trying to make clear not just the benefits but also the obstacles of this general framework.
- From these obstacles, we have formulated a research agenda with the hottest open issues.

All in all, this paper agrees with the observation in [44]: "...MDD has a chance to succeed in the realm of large, distributed, industrial software development, but it is far from a sure bet". We hope that this paper contributes to boost the MDD adoption by practitioners and the design of more powerful MDD methods and better software production processes, and thus increases the likelihood of this bet.

REFERENCES

[1] M. Glinz. "On Non-Functional Requirements". RE 2007.

[2] F.P. Brooks. "No Silver Bullet – Essence and Accidents of Software Engineering". *Computer IEEE* 20(4), 1987.

[3] A. Finkelstein, J. Dowell. "A Comedy of Errors: the London Ambulance Service Case Study". IWSSD 1996.

[4] N. Yusop, D. Zowghi, D. Lowe. "The Impacts of NonFunctional Requirements in Web System Projects". *Int. J. Value Chain Management* 2(1), 2008.

[5] C. Atkinson, T. Kuhne. "Model-Driven Development: A Metamodeling Foundation". *IEEE Software* 20(5), 2003.

[6] S.J. Mellor, A.N. Clark, T. Futagami. "Model-Driven Development". *IEEE Software* 20(5), 2003.

[7] B.H.C. Cheng et al. "Software Engineering for Self-Adaptive Systems: A Research Roadmap". In *Software Engineering for Self-Adaptive Systems*, LNCS 5525, Springer, 2009.

[8] The OMG. "MDA Guide Version 1.0.1". At http://www.enterprise-architecture.info/Images/MDA/MDA%20Guide%20v1-0-1.pdf, 2003.

[9] P. Krutchen. "Architectural Blueprints—The 4+1 View Model of Software Architecture" *IEEE Software* 12(6), 1995.

[10] S. Ceri, A. Bongio, P. Fraternali. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2002.

[11] P. Clements, R. Kazman, M. Klein. *Evaluating Software Architectures. Methods and Case Studies*. Addison-Wesley, 2002.

[12] S.J. Mellor, M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley, 2002.

[13] B. Kitchenham. "Procedures for Performing Systematic Reviews". Keele University Technical Report TR/SE-0401, 2004.

[14] A. Fatwanto, C. Boughton. "Analysis, Specification and Modeling of Non-Functional Requirements for Translative Model-Driven Development". ICCIS 2008.

[15] H. Wada, J. Suzuki, K. Oba. "A Model-Driven Development Framework for Non-functional Aspects in Service Oriented Architecture". *Int. J. of Web Services Research* 5, 2008.

[16] L. Zhu, Y. Liu. "Model Driven Development with Non-Functional Aspects". EA @ ICSE 2009.

[17] The OMG. "UML Profile for MARTE, Beta 2". 2008.

[18] The OMG. "UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, v1.1". 2008.

[19] L. Gonczy, Z. Deri, D. Varro. "Model Transformations for Performability Analysis of Service Configurations". MODELS 2008 Workshops and Symposia.

[20] S. Kugele, W. Haberl, M. Tautschnig, M. Wechs. "Optimizing Automatic Deployment Using Non-functional Requirement Annotations". ISoLA 2008.

[21] F. Molina, A. Toval. "Integrating Usability Requirements that can be Evaluated in Design Time into Model Driven Engineering of Web Information Systems". *Advances in Engineering Software* 40(12), 2009.

[22] A. Solberg, J. Oldevik, J. Aagedal. "A Framework for QoS-aware Model Transformation using a Pattern-based Approach". DOA 2004.

[23] A. Sterritt, V. Cahill. "Customisable Model Transformations based on Non-functional Requirements". IEEE Congress on Services 2008.

[24] S. Röttger, S. Zschaler. "Model-Driven Development for Non-functional Properties: Refinement Through Model Transformation". <<UML>> 2004.

[25] D. Ardagna, C. Ghezzi, R. Mirandola. "Rethinking the use of Models in Software Architecture". QoSA 2008.

[26] S. Gallotti, C. Ghezzi, R. Mirandola, G. Tamburrelli. "Quality Prediction of Service Compositions through Probabilistic Model Checking". QoSA 2008.

[27] G. Rodrigues, D. Rosenblum, S. Uchitel. "Reliability Prediction in Model-driven Development". MoDELS 2005.

[28] V. Cortellessa, A. Di Marco, P. Inverardi. "Non-Functional Modeling and Validation in Model-Driven Architecture". WICSA 2007.

[29] B. Nuseibeh. "Weaving Together Requirements and Architecture". *Computer IEEE* 34(3), 2001.

[30] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer, 2000.

[31] P. Grünbacher, A. Egyed, N. Medvidovic. "Reconciling Software Requirements and Architectures with Intermediate Models". *SoSyM* 3(3), 2004.

[32] J.P. Carvallo, X. Franch, C. Quer. "Managing Non-Technical Requirements in COTS Components Selection". RE 2006.

[33] D. Gross, E. Yu. "From Non-Functional Requirements to Design through Patterns". *Requirements Engineering Journal* 6(1), 2001.

[34] L. Bastos, J. Castro. "Systematic Integration Between Requirements and Architecture". SELMAS 2004.

[35] R.C. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, A. Jansen. "Architectural Knowledge: Getting to the Core". QoSA 2007.

[36] S. Renault, O. Mendez, X. Franch, C. Quer. "A Pattern-based Method for building Requirements Documents in Call-for-tender Processes. *Int. J. of Computer Science & Applications* 6(5), 2009.

[37] J. Horkoff, E. Yu. "Qualitative, Interactive, Backward Analysis of *i** Models". i* Workshop 2008.

[38] D. Roman et al. "Web Service Modeling Ontology". *Applied Ontology Journal*, 1(1), 2005.

[39] D. Ameller, X. Franch. "Usage of Architectural Styles and Technologies in IT Companies and Organizations". EASA 2009.

[40] B. Nuseibeh, S. Easterbrook. "Requirements Engineering: A Roadmap". ICSE 2000.

[41] V. Clerc, P. Lago, H. v. Vliet. "The Architect's Mindset". QoSA 2007.

[42] M. Shaw, D. Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

[43] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture: a System of Patterns*. Wiley, 1996.

[44] B. Hailpern, P. Tarr. "Model-driven Development: The Good, the Bad, and the Ugly". *IBM Systems Journal* 45(3), 2006.